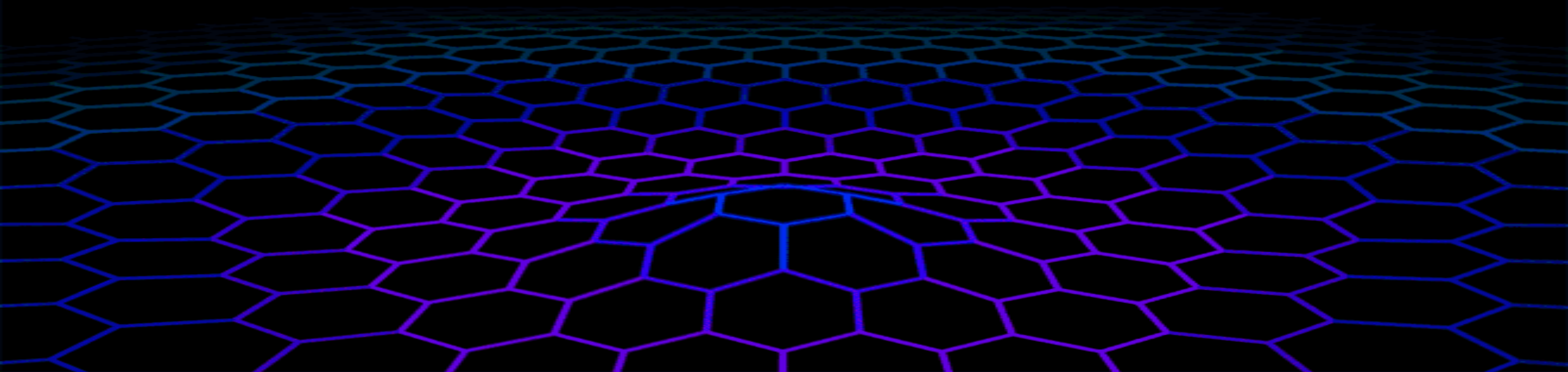
Unisenac
Campus Pelotas

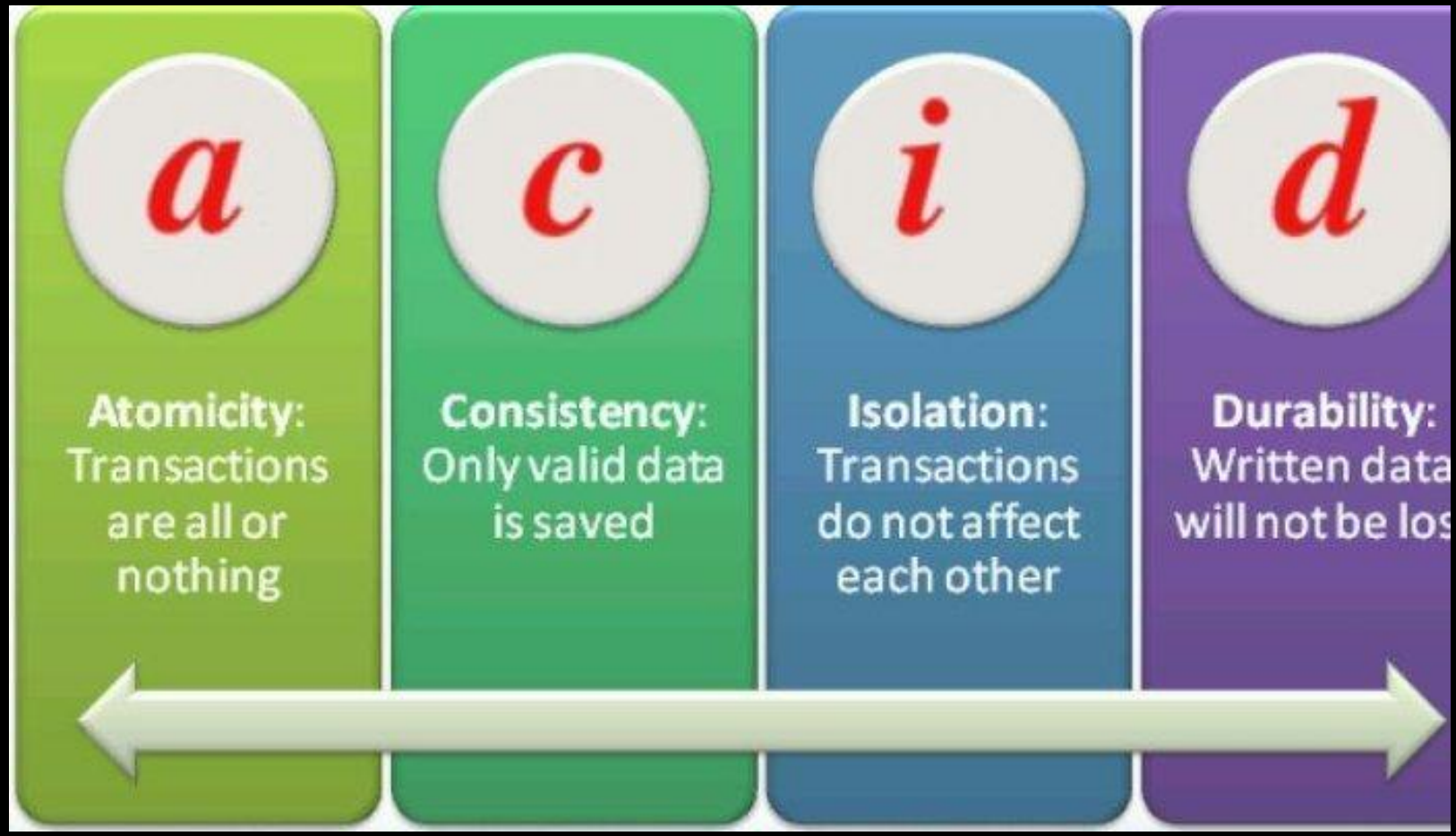
Banco de Dados 2



ACID

ACID

Atomicidade
Consistência
Isolamento
Durabilidade



Propriedades fundamentais de transações em um SGBD.

ACID
Atomicidade

```
/* Uma transação deve ser feita por completo ou ser completamente
revertida. */
```

Exemplo de Atomicidade

```
/* Em uma transação realizamos:
Inclusão de um cliente novo
Geração de uma nota fiscal
Baixa no estoque do produto vendido

Se falhar qualquer parte, nada deve ser gravado. */
```

ACID
Consistência

```
/* Garante que o banco permaneça íntegro antes e após a transação. */
```

Exemplo de Consistência

```
/* O cliente de um banco possui um saldo de R$ 50,00
O cliente não tem limite de crédito (não pode ficar negativo)

Se a transação for uma retirada de R$ 60.00, ela não pode ser
concluída pois a consistência do banco de dados não estaria garantida
deixando a conta com um saldo negativo.


Cliente com R$50 não pode sacar R$60.
Banco deve rejeitar e manter consistência. */
```

ACID
Isolamento

```
/* Transações simultâneas não devem se interferir entre si. */
```

Exemplo de Isolamento

```
/* Duas transações são iniciadas
Ambas estão ligadas diretamente ao mesmo registro no banco de dados
A primeira atualizando
A segunda consultando

O isolamento nos garantirá que a transação de consulta somente será
executada após a transação de atualização ser completada


A transação B só vê o valor atualizado se A fizer COMMIT. */
```

ACID
Durabilidade

/* Garante que a informação gravada no banco de dados dure de forma imutável até que alguma outra transação de atualização, ou exclusão afete-a.

Garante que os dados não sejam corrompidos, ou seja, desapareçam ou se modifiquem sem motivo aparente.

Uma vez feito COMMIT, os dados permanecem salvos, mesmo com falhas no sistema. */

Transações

/* Uma transação é um meio de executar uma ou mais instruções SQL com uma única unidade de trabalho, onde todas ou nenhuma das instruções são bem sucedidas.

Se todas as instruções foram concluídas sem erros, você poderá registrar estas em definitivo no banco de dados.

Caso ocorra algum erro, você poderá retornar a um ponto de salvamento (*savepoint*) ou efetuar um *rollback*, cancelando a alteração. */

Início de uma Transação

```sql
BEGIN;

-- ou

START TRANSACTION;
```

Fim de uma Transação

```
COMMIT;
```

Cancelamento de uma Transação

```
ROLLBACK;
```

Pontos de salvamento: SAVEPOINT

```
SAVEPOINT ponto1;
```

Retornar até o SAVEPOINT

```
ROLLBACK TO ponto1;
```

Desativa o autocommit (MySQL)

```sql
SELECT @@autocommit;  -- Verifica o autocommit

SET autocommit = 0;  -- Desativa o autocommit

SELECT @@autocommit;  -- Verifica o autocommit
```

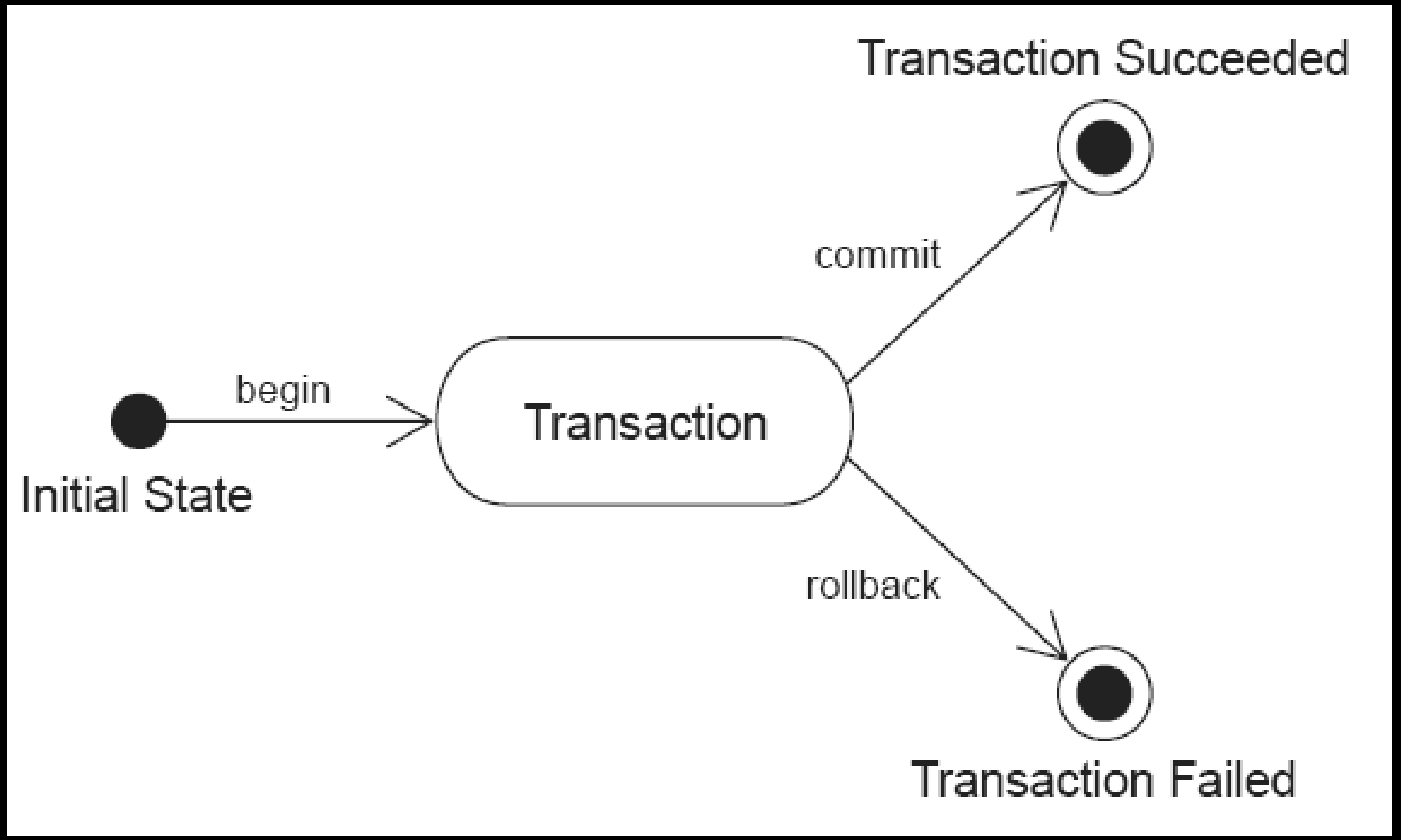
Ativa o autocommit (MySQL)

```sql
SELECT @@autocommit;  -- Verifica o autocommit

SET autocommit = 1;  -- Ativa o autocommit

SELECT @@autocommit;  -- Verifica o autocommit
```

Autocommit (MySQL)

```
-- É possível definir o autocommit = 0 de forma global na seção do [mysqld] no arquivo de configuração (my.cnf).

[mysqld]
autocommit = 0
```

Transações


Transaction Succeeded
commit
begin
Transaction
Initial State
rollback
Transaction Failed

Fenômenos


UMBRO
PIRELLI


THE
BEST


ASSIM
ASSIM
ASSIM
oozzano
oozzano
oozzano
neo
química
GENÉRICOS


Kappa


SIEMENS
mobile

Fenômenos

```
/*

Dirty Read (leitura suja): lê dados não confirmados

Nonrepeatable Read (leitura não repetível): reconsulta traz valor
diferente

Phantom Read (leitura fantasma): nova linha aparece em reconsulta

*/
```

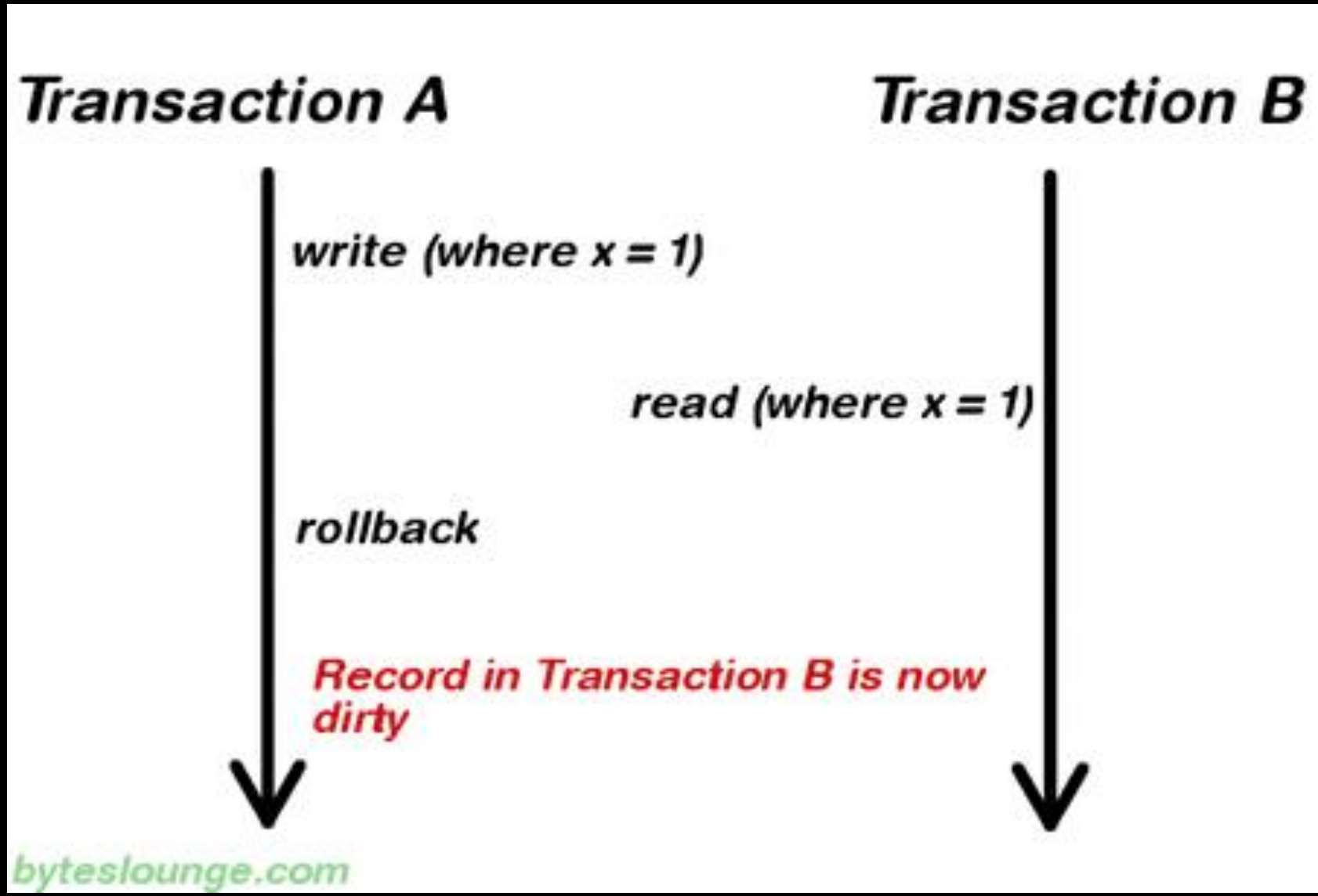
Dirty read (leitura suja)

```
/*
Suponhamos que a transação “A” modifique algum campo da tabela, porém
que ainda não o tenha “commitado”.

Se uma transação “B” efetua um SELECT neste campo e vê o valor
modificado pela transação “A” sem ter o commit efetuado, essa é uma
leitura suja.
*/
```

Dirty read (leitura suja)

```
/*
Isso é um problema em ambientes de tomada de decisão, Relatórios entre
outros.

Caso a transação “A” sofra um Rollback, a transação “B” não saberá
disso e já terá informado o valor “errôneo”.
*/
```

Dirty read (leitura suja)



Nonrepeatable read (leitura não repetível)

```
/*
Ocorre quando um SELECT(leitura) reproduz resultados diferentes quando
ela é repetida posteriormente na mesma transação.
*/
```

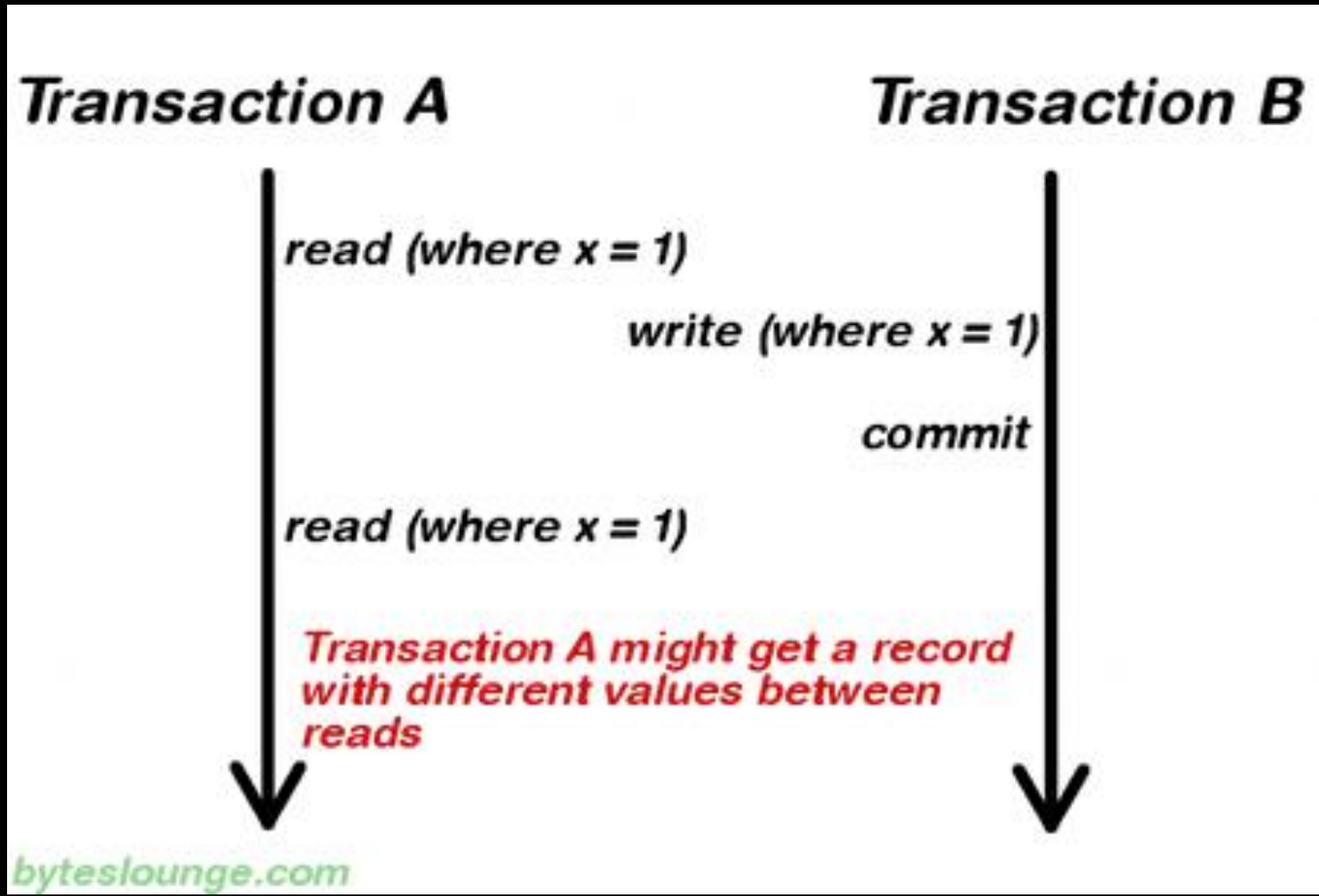
Nonrepeatable read (leitura não repetível)

```
/*
A transação “A” lê o valor de um campo.
Outra transação “B” pode atualizar este valor e “commitá-lo” no banco.
Caso a transação “A” volte a consultar o mesmo campo, ela trará o
valor “comitado” pela transação “B”, ou seja, trata valores diferentes
do mesmo campo na mesma transação executada.
*/
```

Nonrepeatable read (leitura não repetível)

```
/*
Isso é grave caso você opte por alterar algum registro com a condição
de um campo x, onde ele pode assumir um valor y em seguida.
*/
```

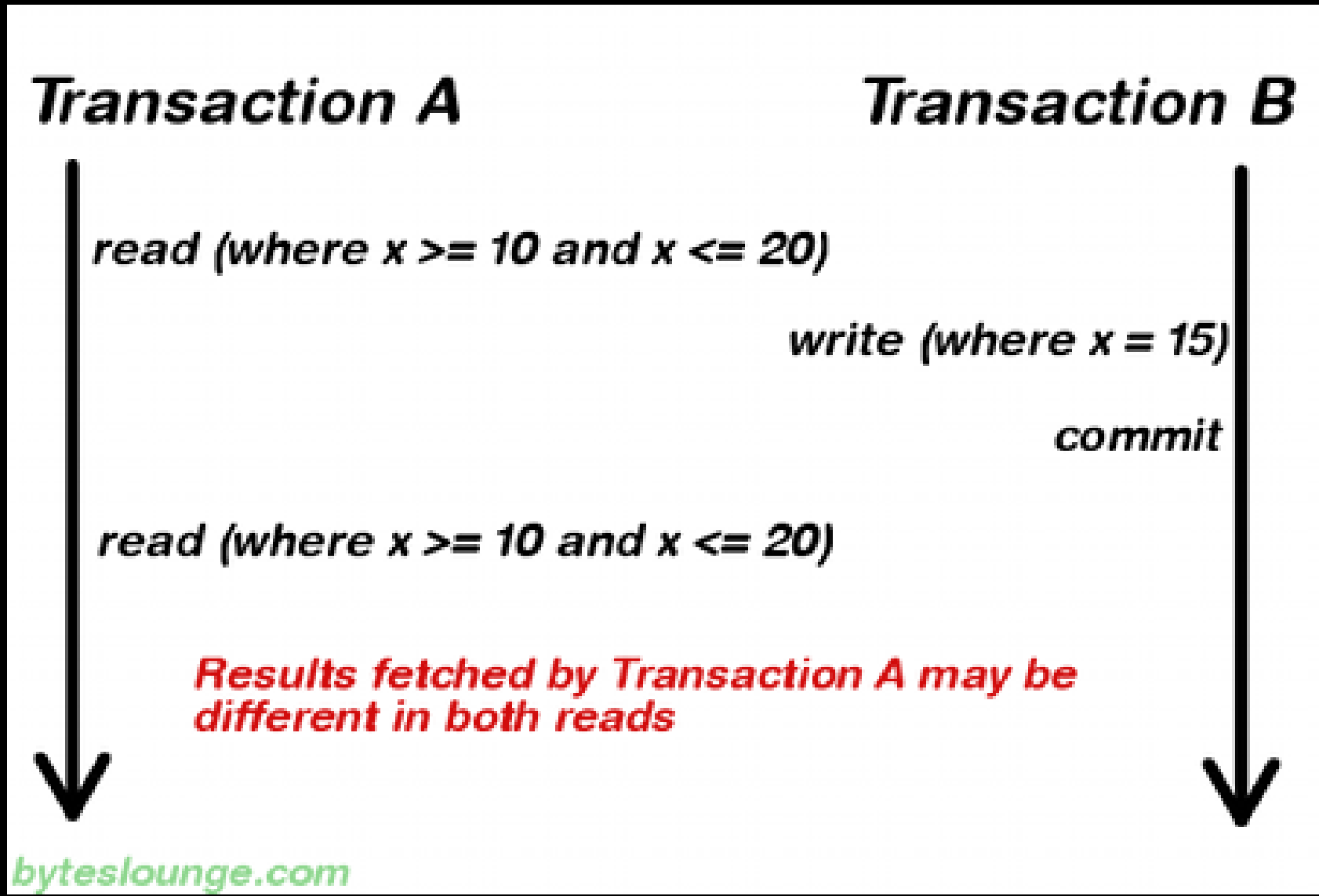
Nonrepeatable read (leitura não repetível)



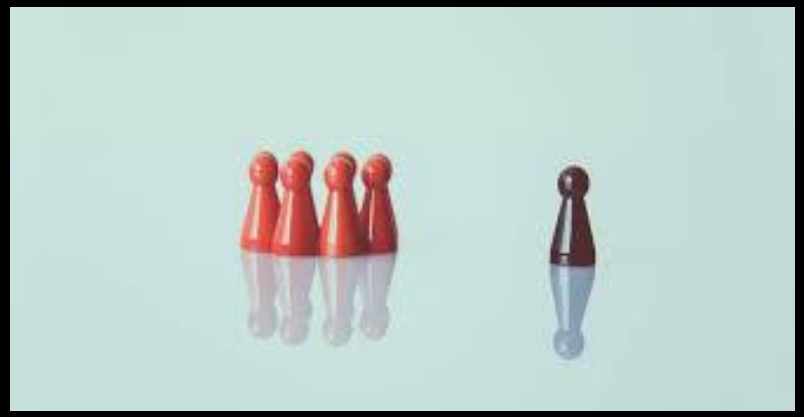
Phantom read (leitura fantasma)

```
/*
Uma transação “A” pode ler um conjunto de linhas de uma tabela com
base em alguma condição WHERE SQL.

Suponhamos que a transação “B” insira uma nova linha que também
satisfaz a clausula WHERE na tabela utilizada pela transação A.

Se a transação “A” for repetida ela verá um fantasma, ou seja, uma
linha que não existia na primeira leitura utilizando a clausula WHERE.
*/
```

Phantom read (leitura fantasma)



Níveis de Isolamento



Níveis de Isolamento SQL

```
-- READ UNCOMMITTED:
-- Permite ler dados não confirmados por outras transações (leitura suja).

-- READ COMMITTED:
-- Só permite ler dados já confirmados (evita leitura suja, mas permite leitura não repetível).

-- REPEATABLE READ (default o MySQL):
-- Garante que as leituras do mesmo dado sejam sempre iguais (evita leitura suja e não repetível, mas permite leitura fantasma).

-- SERIALIZABLE:
-- Isola totalmente as transações, como se fossem executadas em série (evita todos os fenômenos).
```

Consultar nível de isolamento

```
- MySQL:
SELECT @@transaction_isolation;

-- PostgreSQL:
SHOW TRANSACTION ISOLATION LEVEL;
```

Alterar nível de isolamento

```
-- MySQL e PostgreSQL:
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
-- Deve ser executado antes de BEGIN ou START TRANSACTION.

-- Para definir o padrão global no MySQL (arquivo de configuração my.cnf):
mysqld]
transaction-isolation = 'REPEATABLE-READ'   -- ou outro nível desejado
```

Comparativo dos níveis de isolamento e fenômenos

```
/*
+----------------------+--------------+-----------------------+-----------------+
| Nível de Isolamento | Dirty Read   | Nonrepeatable Read   | Phantom Read    |
+----------------------+--------------+-----------------------+-----------------+
| READ UNCOMMITTED    | Possível     | Possível              | Possível        |
| READ COMMITTED      | Impossível   | Possível              | Possível        |
| REPEATABLE READ     | Impossível   | Impossível            | Possível        |
| SERIALIZABLE        | Impossível   | Impossível            | Impossível      |
+----------------------+--------------+-----------------------+-----------------+
*/

-- Legenda:
-- Dirty Read: Leitura de dados não confirmados (transações abertas)
-- Nonrepeatable Read: Mesmo SELECT retorna valores diferentes
-- Phantom Read: SELECT com mesma cláusula WHERE retorna mais linhas
```

```sql
-- Banco para testes (MySQL)
DROP DATABASE IF EXISTS aula08;
CREATE DATABASE aula08;
USE aula08;

CREATE TABLE ator (
    id   INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL
);

INSERT INTO ator (nome) VALUES
  ('Adam Sandler'),
  ('Hey Decio'),
  ('Jamie Foxx'),
  ('Joaquim Phoenix'),
  ('Jude Law'),
  ('Meryl Streep'),
  ('Michael Douglas'),
  ('Tom Cruise');
```

```sql
-- Banco para testes (PostgreSQL)
DROP DATABASE IF EXISTS aula08;
CREATE DATABASE aula08;
\c aula08;

CREATE TABLE ator (
    id   SERIAL PRIMARY KEY,
    nome VARCHAR(100) NOT NULL
);

INSERT INTO ator (nome) VALUES
  ('Adam Sandler'),
  ('Hey Decio'),
  ('Jamie Foxx'),
  ('Joaquim Phoenix'),
  ('Jude Law'),
  ('Meryl Streep'),
  ('Michael Douglas'),
  ('Tom Cruise');
```

-- Transação com ROLLBACK

```
-- MySQL:
SET autocommit = 0;
START TRANSACTION;
DELETE FROM ator;
SELECT * FROM ator;
INSERT INTO ator (nome) VALUES ('Will Smith');
SELECT * FROM ator;
ROLLBACK;
SELECT * FROM ator;

-- PostgreSQL:
BEGIN;
DELETE FROM ator;
SELECT * FROM ator;
INSERT INTO ator (nome) VALUES ('Will Smith');
SELECT * FROM ator;
ROLLBACK;
SELECT * FROM ator;
```

-- Transação com COMMIT

```
Transação com COMMIT

-- MySQL:
START TRANSACTION;
DELETE FROM ator;
INSERT INTO ator (nome) VALUES ('Will Smith');
COMMIT;
SELECT * FROM ator;

-- PostgreSQL:
BEGIN;
DELETE FROM ator;
INSERT INTO ator (nome) VALUES ('Will Smith');
COMMIT;
SELECT * FROM ator;
```

```sql
-- SAVEPOINT e ROLLBACK TO

START TRANSACTION;
INSERT INTO ator (nome) VALUES ('Bruce Willis');
SAVEPOINT antes_do_erro;
-- Próxima linha gera erro
INSERT INTO ator (nome) VALUES (NULL);
ROLLBACK TO antes_do_erro;
INSERT INTO ator (nome) VALUES ('Tom Hanks');
COMMIT;
SELECT * FROM ator;
```

-- Stored Procedure com controle de erro (MySQL)

```
DELIMITER $$
CREATE PROCEDURE insere_atores()
BEGIN
  DECLARE erro BOOL DEFAULT FALSE;
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET erro = TRUE;

  START TRANSACTION;
    INSERT INTO ator (nome) VALUES ('Angelo Luz');
    INSERT INTO ator (nome) VALUES ('Pablo Escobar');
    INSERT INTO ator (nome) VALUES (NULL);
  IF NOT erro THEN
    COMMIT;
    SELECT 'Transação OK' AS resultado;
  ELSE
    ROLLBACK;
    SELECT 'Erro na transação' AS resultado;
  END IF;
END$$
DELIMITER ;

CALL insere_atores();
```

-- SIGNAL SQLSTATE no MySQL

```
DELIMITER $$

CREATE PROCEDURE inserir_ator(IN nome_ator VARCHAR(100))
BEGIN
  IF nome_ator IS NULL OR LENGTH(TRIM(nome_ator)) = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Nome do ator não pode ser vazio';
  END IF;

  INSERT INTO ator (nome) VALUES (nome_ator);
END$$

DELIMITER ;




CALL inserir_ator('Tom Hanks'); -- Funciona
CALL inserir_ator('');          -- Erro: Nome do ator não pode ser vazio
```

-- SIGNAL SQLSTATE no MySQL

```
DELIMITER $$

CREATE FUNCTION inserir_ator(nome_ator VARCHAR(100))
RETURNS VARCHAR(100)
DETERMINISTIC
BEGIN
  IF nome_ator IS NULL OR LENGTH(TRIM(nome_ator)) = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Nome do ator não pode ser vazio';
  END IF;

  INSERT INTO ator (nome) VALUES (nome_ator);
  RETURN 'Ator inserido com sucesso';
END$$

DELIMITER ;

SELECT inserir_ator('Tom Hanks'); -- Funciona
SELECT inserir_ator('');          -- Erro: Nome do ator não pode ser vazio
```

-- SIGNAL no PostgreSQL (RAISE EXCEPTION)

```
CREATE OR REPLACE PROCEDURE inserir_ator(nome_ator TEXT)
LANGUAGE plpgsql
AS $$
BEGIN
  IF nome_ator IS NULL OR LENGTH(TRIM(nome_ator)) = 0 THEN
    RAISE EXCEPTION USING
      MESSAGE = 'Nome do ator não pode ser vazio',
      ERRCODE = 'P2001';
  END IF;

  INSERT INTO ator (nome) VALUES (nome_ator);
END;
$$;
```

```
CALL inserir_ator('Tom Hanks'); -- Funciona
CALL inserir_ator('');          -- ERRO: Nome do ator não pode ser vazio
```

-- SIGNAL no PostgreSQL (RAISE EXCEPTION)

```sql
CREATE OR REPLACE FUNCTION inserir_ator(nome_ator TEXT) RETURNS VOID AS $$
BEGIN
  IF nome_ator IS NULL OR LENGTH(TRIM(nome_ator)) = 0 THEN
    RAISE EXCEPTION USING
      MESSAGE = 'Nome do ator não pode ser vazio',
      ERRCODE = 'P2001'; -- Código personalizado de erro (usuário)
  END IF;

  INSERT INTO ator (nome) VALUES (nome_ator);
END;
$$ LANGUAGE plpgsql;
```

```sql
SELECT inserir_ator('Tom Hanks');  -- Funciona
SELECT inserir_ator('');           -- Erro: Nome do ator não pode ser vazio
```

```
-- SQLSTATE Exemplos de códigos padrão ANSI

/*
  23000 - Violação de integridade referencial (ex: chave duplicada ou nula)
  45000 - Erro personalizado definido com SIGNAL (MySQL)
  42000 - Erro de sintaxe SQL
  HY000 - Erro genérico no MySQL
  P0001 - Exceção definida pelo usuário no PostgreSQL (RAISE EXCEPTION padrão)
  P2001 - (Exemplo personalizado) Erro definido por RAISE EXCEPTION com ERRCODE

Use SIGNAL no MySQL e RAISE EXCEPTION no PostgreSQL para criar regras
de validação e mensagens de erro amigáveis.
*/
```

Script 1

O arquivo `BD2-A08-parte1.SQL` contém um script com instruções para exercitar.

Script 2

Execute, passo-a-passo, as instruções contidas no arquivo `BD2-A08-parte2.SQL`

Script 3

O arquivo `BD2-A08-parte3.SQL` contém uma atividade suculenta e saborosa.

Ainda faltam slides (não te apressa)

Backup e Restore em MySQL e PostgreSQL

BackUp com MySQL Dump

-- Backup com mysqldump

Ferramenta oficial do MySQL para gerar backups no formato SQL.
Gera um arquivo de texto contendo comandos CREATE TABLE e INSERT.

Exemplo de backup:

```
mysqldump -h localhost -u root -p nomedobanco > "d:/backup.sql"
```

-h: host (padrão é localhost)
-u: usuário (ex: root)
-p: solicitar senha
nomedobanco: nome do banco de dados
> backup.sql: redireciona a saída para um arquivo

Restore no MySQL

-- Restauração (restore) no MySQL

```
Restauração (restore) no MySQL:

mysql -u root -p nomedobanco < d:/backup.sql

Se já estiver dentro do cliente MySQL:
SOURCE d:/backup.sql;
```

BackUp com PostgreSQL pg_dump

```
-- Backup com pg_dump

-- Ferramenta oficial do PostgreSQL para gerar backups em diferentes
formatos.
-- Exemplo de backup em formato SQL:
pg_dump -U postgres -d nomedobanco > "d:/backup.sql"

-U: usuário do PostgreSQL
-d: nome do banco de dados
> backup.sql: redireciona a saída para um arquivo SQL

Exemplo de backup em formato personalizado (compactado):
pg_dump -U postgres -F c -f "d:/backup.dump" nomedobanco

Opções:
-F c: formato custom (binário), ideal para uso com pg_restore
-f: define o nome do arquivo de saída
```

Restore com pg_restore

-- Restauração (restore) no PostgreSQL

```
Arquivo .sql (formato texto):
psql -U postgres -d nomedobanco < d:/backup.sql

Se já estiver conectado ao PostgreSQL:
\i d:/backup.sql

Arquivo .dump (formato binário):
pg_restore -U postgres -d nomedobanco d:/backup.dump
```


Sabe onde está o exercício?
Lá no **BlackBoard**